Setting up GCloud SDK

Step 1 : Sign up at https://cloud.google.com/ and enable billing.

Step 2 : Navigate to the Compute Engine from the left menu bar and create a Virtual Machine.

The following steps are to be done **after creating a vm-instance in GCP**. This is to prepare the instance for hosting your web application.

Setting up GCloud SDK in your system

https://cloud.google.com/sdk/docs/quickstarts

After choosing your system and running the commands, go to your vm-instance.

Copy the GCloud command in the SSH dropdown in the VM-instance. Paste the command in your preferred terminal and you are good to go.

In case of 'gcloud auth login'

Type **gcloud auth login** in terminal

Copy the URL provided, and paste it in browser

Choose your google account

Copy the verification link and paste it in the terminal

Steps to do with newly created Linux instance:

You should be logged in to your remote instance

- Creating a root password -> **sudo passwd**
- Login as admin -> **su**
- Go to root -> **cd**
- Update the packages -> **apt upgrade && apt update**
- Install necessary packages ->
    - **apt install nodejs**
    - **apt install npm**
- Check node version ->
    - **node -v**
    - **npm -v**
- Update the node version ->
    - **npm i n -g**
    - **n latest** or **n stable** (depending on what you want)
- Exit the shell to check the update node/npm version ->
    - **exit**
    - **node -v** (should now show the latest version)
    - **npm -v** (should now show the latest version)
- Pull your node project from Github or create your project on the server.

VPC Network Firewall Rules

Enabling PORT number from VPC network (Test Purpose until we implement Reverse Proxying)

- Go to VPC network in GCP console
- Under Firewall rules, click on default-allow-http
- CLick on Edit on top
- Then, in the bottom, under protocols and port add your app's PORT number.
    - tcp:80,3000 (Here the Test App's port number is 3000)
- Run your app from terminal -> npm start (whatever your command is)
- Now type in your VM IP address in the browser, with your PORT number. Your app should be running.

DEPLOYING NODE APP IN PRODUCTION

I) Using 'nohup' as an Alternate to PM2

A linux command which runs processes in the background. Refer your Linux Reading material in Week 2 for more info on the nohup command.

```
nohup python -m SimpleHTTPServer <PORT> >> log.txt &
```

The above command will run the server in the background and also keep the logs in a file called log.txt.

To start your node server, simply change the python command to a node server command

II) Installing PM2

To run your node app in the background, we need to install a package called **PM2**

- Install PM2 from npm -> **npm i pm2**
- Type -> **pm2 list**

  { this will show the apps pm2 is running }
- Instead of npm start, type -> **pm2 start** <filePath>

Now your app is running in the background.

Other commands with **pm2**

- **pm2 list** (to see the apps pm2 is running)
- **pm2 start** <filePath> (to start an app)
- **pm2 stop** <index of the app> (to stop the app)
- **pm2 restart** <index> (to restart the app)

FIREWALLS

OS level firewall

IP Tables

https://www.linode.com/docs/security/firewalls/control-network-traffic-with-iptables/

UFW : uncomplicated firewall

https://help.ubuntu.com/community/UFW

REVERSE PROXYING WITH NGINX

First check if your app is online

- **pm2 list**

Check if your app is online and then visit your IP with the port number of your app to see if it is online.

However, it's not a good practice to give access via the port number of the app. To prevent this, we need to implement **reverse-proxying.**

All the traffic to VM should only be available via 80 (HTTP) and 443 (HTTPS) ports.

That means Port 443 and Port 80 should reverse proxy the traffic to the port of your node app.

In addition, it's not a good practice to give access to VM via the IP address, it should be available via the Domain Name. Either drop accessing the VM with IP or redirect it to domain.

Note : Configure Domain Name for your VM to proceed further with Reverse Proxy

SETTING UP CLOUD DNS

- Go to the **Cloud DNS** section under **Network Services**
- Click on '**Create a DNS Zone**'
    - Make zone type **public**
    - Add a Zone name (can be anything)
    - Keep DNSSEC off and hit create
    - Now you have to add a record set for the Domain
    - Click on '**Add Record set**'
    - Add your VMInstance's **IPv4 address** in the IP section (leave everything else empty) and create.
    - After this add another record set with **www** in the DNS Name section, add the IP address again and hit create.
- Now wait for about 10 to 15 mins and then check with your <**domain-name**>:**PORT** number (as of now we haven't enabled reverse proxying). You should be able to see your app.

Now we will get back to **Reverse Proxying with NGINX**

How to proxy the traffic from PORTs 80 & 443 to your App's PORT ?

- First just do (just for safety) ->
    - **apt update && apt upgrade**
- Now do -> **apt install nginx**

- After installation, check the status of NGINX
    - **systemctl status nginx**
    - The Active should show active (running)
- If it is not active, you need to do ->
    - **systemctl start nginx**
- Now check your app again -> it should show an nginx page, unless you have put an html file in the **/var/www/html** Path.

SETTING UP NGINX SERVER BLOCKS

By Default all web servers will point out to **/var/www/html**

To override this rule to custom a path, we have to create a server block.

Here our new path of web server block will be **/var/www/<domain.com>/html**

- **mkdir -p /var/www/<domain>.com/html**
- **nano index.html**
- From the earlier path (**/var/www/<domain>.com/html**)
    - **nano /etc/nginx/sites-available/<domain-name>**
    - (this will create a new file and open in nano mode)

The above created file is to configure the nginx server block to redefine the default path **/var/www/html**

Now write the following in the file with **nano** :

```
server {
  listen 80;
  listen [::]:80;
  root /var/www/<example>.com/html;
  index index.html index.htm index.nginx-debian.html;
  server_name <example>.com www.<example>.com;
  location / {
                  try_files $uri $uri/ =404;
  }
}
```

Now add a soft link from **sites-enabled** folder to **sites-available**

```
ln -s /etc/nginx/sites-available/<domain-name> /etc/nginx/sites-enabled/
```

Now save the file. We also need to test the file for possible errors. For that

- **nginx -t**

- If everything went well, the reply should say -> test is successful. If not, re-check what you wrote in the file.
    - Then **systemctl restart nginx**

- Now check your domain again. If the server block is working properly, now it should show your app's page.

(Now you have overridden the web server's path to a custom path which points to your app.)

Reverse Proxying Node app with NGINX

- **nano etc/nginx/sites-available/<domain-name>**
- We have to put the location tag in the file that we wrote earlier. Now write this in the location tag

```
location / {
         proxy_pass http://localhost:3000;
         proxy_http_version 1.1;
         proxy_set_header Upgrade $http_upgrade;
         proxy_set_header Connection 'upgrade';
         proxy_set_header Host $host;
         proxy_cache_bypass $http_upgrade;
       }
```

Now you can remove the **PORT 3000** (or whatever your PORT is) from the Firewall Rules section of the VPC networks.

Now your app is only accessible via the domain, without any port. It won't be accessible via the PORT number.

Running Multiple Node Apps from one Instance

Multiple apps can be run on one VM-instance only if they are running on different PORT numbers. So your second node app should be running on a different PORT number than your first one.

- Go to /bin/www (express-generator) or Deploy your another node app in the server as **root**

Now another thing to remember is that the routes of the two apps should not conflict with each-other.

So modify the route of your new node app (so that it is different from that of the first app.)

After that all we need to do is to reverse proxy traffic from the new URL to the second node app. For that we need to add another location block.

```
location /<new-route> {
        proxy_pass http://localhost:<new-PORT>;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
      }
```

Your previous app is already running on the path **/**

So you need to provide a new path for your second app /<your-path>

- Save the document
- **nginx -t**
- If test is successful : **systemctl restart nginx**

Now you can check your domain/<new-path> it should take you to your second node app.

IMPLEMENTING SSL (SECURE SOCKET LAYER)

Go to certbot.eff.org

Choose your server operating system, for me it is ->

- Running NGINX on Ubuntu 18.04/20 lts (bionic)

Login to your cloud CLI as root.

- **apt-get update**
- **apt-get upgrade**
- **add-apt-repository universe**
- **add-apt-repository ppa:certbot/certbot**
- **apt-get update**
- **apt-get install certbot python-certbot-nginx**
- **certbot --nginx**

Follow the instructions on the command-line.

You can view the nginx configuration file to view what changes certbot made to it.

Setting up a crontab to automatically renew SSL certificates

Run the following command to renew the certificate

- **certbot renew --dry-run**

Redirecting IP to Domain

```
server {
    listen 80;
    server_name <IP address>;
    return 301 https://<domain-name>$request_uri;
}
```

We need to add this server block to the end of the nginx configuration file to redirect IP to domain.

Blocking IP address (dropping IP address access)

```
server {
    listen 80;
    server_name <IP address>;
    return 444;
}
```

Add the following at the end of the nginx config file to drop the IP.

Setting up Sub-Domains

- Go to Cloud DNS under Network Services.
- Add a record set to the already existing Zone.
- Add your subdomain (app.<domain-name>)

- Keep the IP of your current VM or you can add a new VM for another app on another instance.

REVERSE-PROXYING FOR SUB-DOMAINS

If your IP is from the same VM instance ：

- We need to add another server-block to serve that route.

So go back to the nginx configuration file and add the following configuration

```
server {
  listen 80;
  listen[::]:80;
  root /var/www/<example.com>/html;
  index index.html index.htm index.nginx - debian.html;
  server_name <sub-domain>.<example.com>
  location / {
           proxy_pass http://localhost:<PORT>;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection 'upgrade';
                proxy_set_header Host $host;
                proxy_cache_bypass $http_upgrade;
  }
}
```

- **nginx -t**
- **systemctl restart nginx**

Now you have changed your server_name from <**domain**>.com to

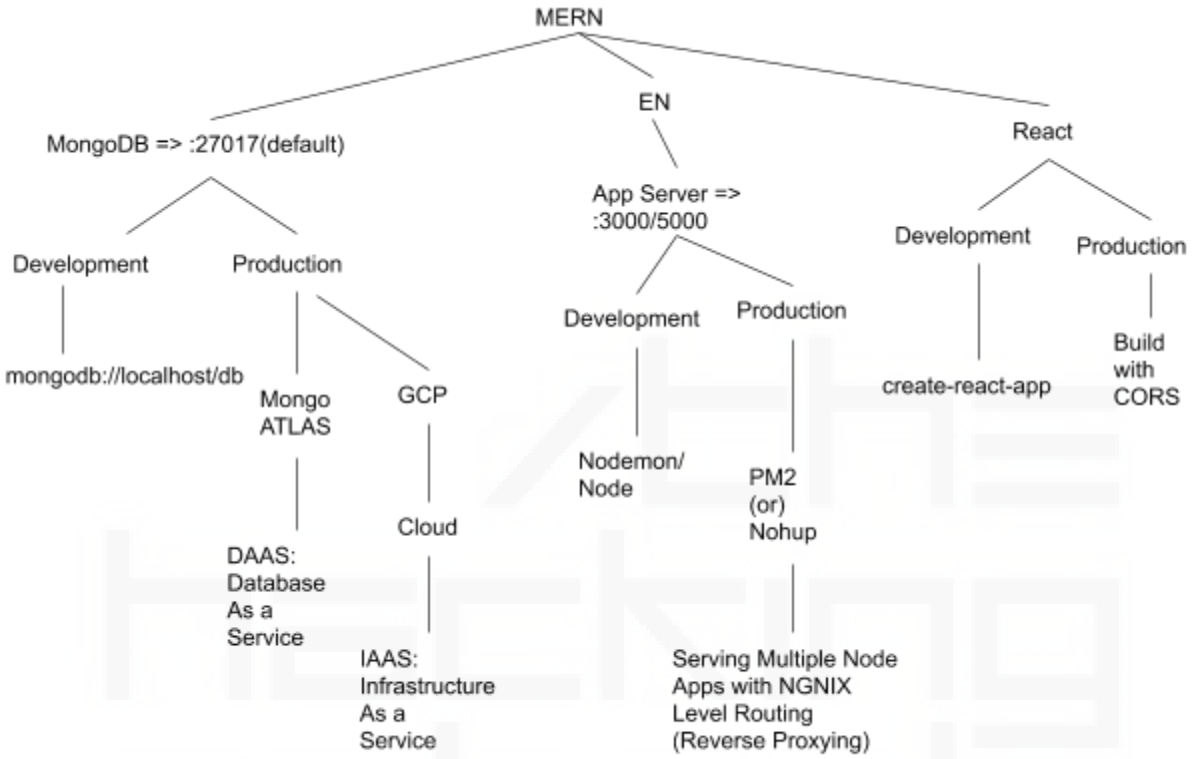
<**subdomain**>.<**domain**>.com.

Now Setup a Node App for sub-domain

- Restart pm2 and then check the access sub-domain

MERN STACK DEPLOYMENT PROCEDURES

Depending on your scaling requirements



CLOUD COMPUTING

IAAS: Infrastructure as a service Eg :Google Compute Engine (GCE)
PAAS: Platform as a service Eg : Heroku
SAAS: Software as a service Eg : Slack etc.,

Depending on the scaling requirements, the following deployment procedures should be followed :

Procedure 1 :
Serve React Build within Express Static of Node-Express app + MongoDB Server *(Server 1)*

Procedure 2 :
Serve React Build in a Apache/Nginx Server while proxying or Enabling CORS in the backend Server (Server 1 for React Build) + Server 2 (Node/Express + MongoDB)

Procedure 3 : Serve React Build in Server 1 + Node/Express in Server 2 + MongoDB in Atlas/Server 3

Procedure 1 (Development Mode)

Project Folder Structure ->

- **Client folder**
    - package.json

```
"scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
```

- **Server folder**
    - package.json

```
 "scripts": {
    "start": "node server.js"
  },
```

Project folder

- package.json

```
"scripts": {
    "react": "npm start --prefix client",
    "node" : "npm start --prefix server",
    "project":"concurrently \"npm run react\" \"npm run node\""
  },
```

Procedure 1 (Production Mode)

- After the development process is complete, we need to deploy the react app.
- From package.json, remove the proxy you might have added while in development
- Making a production build of the react app
    - **npm run build**
- Now you should have got another folder in the react app called build.
- This will contain the optimised build of the entire react app.
- Now we take this folder to the server folder that we already had.
- Now go to app.js of the express app

- Under **const app = express()**
- Write ->
    - **app.use(express.static('build'))**

Procedure 2 :

Serve React Build in a Apache/Nginx Server while proxying or Enabling CORS in the backend Server (Server 1 for React Build) + Server 2 (Node/Express + MongoDB)

CORS : https://www.npmjs.com/package/cors

https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Procedure 3 :

Serve React Build in Server 1 + Node/Express in Server 2 + MongoDB Atlas/Server 3

CONTAINERS | DOCKER

Setting up a Docker Engine

- First step is to create a new VM-instance in GCP and update the packages.
- https://docs.docker.com/install/linux/docker-ce/ubuntu/
- The above is the link to the Docker docs.
- Follow the instructions under **Install using the convenience script**
- To see which images are locally available ->
    - **docker images**
- After this, you can run ->
    - **docker run hello-world**
- To see which all images are running ->
    - **docker ps -a**
- To stop an image ->
    - **docker stop <docker-id>**
- Remove an image
    - **docker rm <docker-id>**
- To restart an image (without creating another instance) ->
    - **docker start --attach <container name>**
- You can check whether or not it started another instance by -> **docker ps -a**

Running ubuntu image

- **docker run -it ubuntu bash**
- We add **-it** to specify that we want to interact with the shell. (to step into the container)
- To download a newer image -> **docker pull ubuntu:18.04**

Composing a Docker

- First make sure docker is installed in the cloud
- Create a node app and clone it to your cloud
- Create a file -> **nano Dockerfile**
- Write the following inside the file ->
    - **FROM node:13**
    - **WORKDIR /app**
    - **COPY package.json /app**
    - **RUN npm install**
    - **COPY . /app**
    - **EXPOSE <**PORT-number**>**
    - **CMD npm start**

- After writing the file ->
    - **docker build ./ -t <tag-name>**

Pushing an Image to Docker-Hub

- Create an account in Docker-hub
- Copy the account/repo-name given by DockerHub ->
    - { docker push **prashanthteja/node**:tagname }
    - Yours too should look like this with your account and repository-name.
- **docker build ./ -t prashanthte/node** in the file where the Dockerfile is
- **docker login**
- Run the docker container in detached mode
    - **docker run -d <image id>**
- **docker commit <container id> prashanthteja/node:<tag>**
- **docker push prashanthteja/node:tagname**

Now refresh the Docker-Hub account, you should see your latest commit.

Binding VM's port with that of the container

- **docker run -p <VM's PORT>:<container-PORT> tag-name**

Resources

https://docker-curriculum.com/

https://stackify.com/docker-tutorial/

JENKINS | DevOps

Implementing Continuous Integration and Continuous Deployment (CI/CD for MERN Stack)

How to setup jenkins in the cloud

https://pkg.jenkins.io/debian-stable/

- **apt-get install default-jdk**
- **wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -**
- **sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'**
- **apt-get update**
- **apt-get install jenkins**
- Check status of jenkins -> **systemctl status jenkins**
- To give root access to jenkins
    - **nano /etc/sudoers**
    - Under **%sudo**.... Add the following ->
    - **jenkins ALL=(ALL) NOPASSWD:ALL**

Sign in to jenkins

Jenkins will always be on port 8080, so make sure you have allowed that port in firewall rules.

- Manage Jenkins
- Go to available tab
- Search **nodejs**
- Check the box
- Install without restart
- Search for github
- Check the box for **github integration**
- Install without restart

How to add a webhook to GitHub to trigger automatic deployment

- Go to your GitHub repository
- Go to **settings**
- Go to the **Webhooks** section.
- Click on **Add a webhook**.
- Copy your Jenkins URL { IP-address:8080 }
- The payload URL should look something like this
    - http://34.93.220.60:8080/github-webhook/
    - { http://<jenkins-url>:8080/github-webhook/ }

    - The slash at the end is very important
- Check the **Just the push event** under **Which events would you like to trigger this webhook?**
- After this, click add webhook and wait for the green tick. If it doesn't come, your GitHub account is not connected with Jenkins.
- On Jenkins page, in the configure page, under **Build Triggers** ->
    - Check the **GitHub hook trigger for GITScm polling** box
    - Hit **Save**
- Now you can push to GitHub and watch your changes go live.